Simulation 08

Programming Techniques · Home Practice

Assessment

Full exam: 6 exercises, 4 + 4 + 2 + 4 + 4 + 2 = 20 points

Partial I: exercises 1·3, 7 + 7 + 6 = 20 points

Partial II: exercises 4·6, 8 + 8 + 4 = 20 points

Instructions

Write independent Python solutions. Use only tools covered in the course. Read the requirements, assumptions and expected results of each exercise before beginning. Input files mentioned by name are included in the accompanying data folder. Each exercise is independent; no solution from another exercise is needed. The reference values illustrate the required behavior, but solutions must work with other valid inputs. No additional validation, formatting or functionality is required beyond what is explicitly requested.

The six exercises follow on the next pages.

Simulation 08

Full exam: 4 + 4 + 2 + 4 + 4 + 2 points. **Partial I:** 7 + 7 + 6. **Partial II:** 8 + 8 + 4.

Exercise 1 – Count matching labels by category (4 points – full exam)

Partial I value: 7 points

A sports association stores a list of event records. Each record is a dictionary containing a category name and a list of text labels. The association wants one count per category: how many of that category's individual labels contain at least the required number of characters.

Each record provides a category and a list of labels. Count only labels meeting the stated length requirement, combining contributions from multiple records belonging to the same category. Categories with no qualifying labels still appear with zero. The result is a category-to-count dictionary, not a list of individual labels.

Required results and operations

a) Process all records and count, separately for each category, the labels containing at least `minimum` characters
b) If a category appears in several records, combine its counts; include categories even if no label qualifies
c) Print each category and its count in alphabetical order
d) Process the individual labels with nested loops rather than counting whole lists by their lengths

Reference example

```
records = [
    {'category': 'A', 'labels': ['red', 'blue', 'green']},
    {'category': 'B', 'labels': ['box', 'large']},
    {'category': 'A', 'labels': ['gold']},
    {'category': 'C', 'labels': []}
]
minimum = 4

Expected output:
A: 3
B: 1
C: 0
```

Exercise 2 – Detect consecutive increases (4 points – full exam)

Partial I value: 7 points

A sequence of daily observations is given. Report which adjacent pairs represent increases.

The question compares adjacent readings, not all possible pairs. Position i qualifies if the reading at i+1 is strictly larger than the reading at i. Preserve the chronological order of qualifying positions and do not count equal consecutive values as increases.

Input and assumptions All observations are numeric; empty and one-element lists have no adjacent increases.

The position refers to the later member of each increasing pair.

Required results and operations

a) Implement increasing_positions(values), returning indexes i where values[i] is greater than values[i-1]
b) Implement count_increases(values) using the same definition of increase
c) Print positions and their count
d) Use zero-based indexing

Tools and restrictions: Use functions with parameters and return values together with loops or list operations from Lectures 1–10; no libraries or file processing.

Reference example

```
measurements=[5,7,6,9,10]

Expected output:
Positions: [1, 3, 4]
Number: 3
```

Exercise 3 – Longest name in nested departments (2 points – full exam)

Partial I value: 6 points

A company directory uses dictionaries and sub-dictionaries to store employee names as values. Return a longest name.

Search the full nested department structure and return the longest string name. The result is the name itself rather than its length. If more than one name has the same maximum length, use the specified tie convention. Empty subgroups are possible and must not cause a failure.

Input and assumptions

The outer object is a dictionary. Each value is either a string or another nested dictionary. Keys are strings; empty dictionaries may occur. If several names have maximum length, keep the first visited; return None for an empty structure.

Required results and operations

a) Define `longest_name(data)` and make it return the result described above for the entire structure
b) Process dictionary values at every nesting level; empty dictionaries must be handled correctly
c) Call the function(s) on the reference data and print the returned result(s) as shown

Tools and restrictions Use ordinary recursion, conditions, loops when processing nested containers, and simple string/list/dictionary operations taught in Lectures 1–10. Do not use flattening libraries or `isinstance()`; `type(x) == list` or `type(x) == dict` is sufficient.

Reference example

```
data = {'lab': {'a': 'Mia', 'b': 'Charlotte'}, 'admin': {'c': 'Oliver'}}

print(longest_name(data))

Expected output:
Charlotte
```

Exercise 4 – Positive and negative readings report (4 points – full exam)

Partial II value: 8 points

An inspection station writes one integer result per line in `values.txt`. The supervisor wants to know how many readings are positive, how many are negative and how many are exactly zero. These categories are mutually exclusive: every valid reading contributes to exactly one count.

Read the file and print the three counts in the order `Positive`, `Negative`, `Zero`. For example, a positive reading increases only the positive count; it does not contribute to the other two. The supplied file contains valid integers, and no additional error handling is required.

Input and assumptions The file exists, all nonempty lines contain valid integers, and there is at least one integer.

Save three labeled counts to summary.txt, in the order Positive, Negative, Zero. Print the same three lines.

Required results and operations

a) Read all integers from values.txt, one integer on each nonempty line
b) Calculate the three requested summary values

c) Save the report in summary.txt and print the same lines

Tools and restrictions: Use ordinary text-file operations and Python numeric comparisons. No pandas, NumPy or exception handling.

Reference example

```
values.txt:
5
10
3
10
8

Expected printed result:
Positive: 5
Negative: 0
Zero: 0
```

Exercise 5 – Correcting a list of sensor readings (4 points – full exam)

Partial II value: 8 points

A text file contains one intended integer per line, although some lines may not represent valid integers. The main program must read and validate the input lines and then pass the entire list of valid numbers to an imported function. The helper transforms the list and returns the ordered results. The main program writes `result.txt` and prints the requested total.

Input and assumptions The text file exists. Preserve the order of valid records and ignore blank lines or invalid integer values as specified.

Required results and operations

a) Create `helpers.py` defining `increase_values(values)`, which loops over all numbers and returns a list of transformed values in the same order
b) In `main.py`, read `source.txt`, skip invalid integer conversions by catching `ValueError` and call the imported function once
c) Write the returned list to `result.txt`, one value per line, and print the sum as in the example

Tools and restrictions: Use basic loops, functions, imports, standard file handling and the named standard-library modules. No external packages are needed.

Reference example

```
Input file:
3
6
3

Expected printed result:
30
```

Exercise 6 – A specialized library item (2 points – full exam)

Partial II value: 4 points

A general library item has a title; an audio recording adds its duration.

The specialized item remains an instance of the general library-item type while extending its behavior. Keep the provided parent methods intact, initialize shared attributes through the parent and display the result of the specialized method on an instance of the child class.

Input and assumptions

No borrowing operations or validation are required. No exception handling or external data files are required.

Required results and operations

a) Define `AudioBook` as a subclass of the provided `LibraryItem`; keep the parent class unchanged
b) Extend the child to store the recording duration, while reusing the parent initialization appropriately
c) Override `.describe()` to extend the inherited description with the additional information in the format shown
d) Create the reference instance and print its resulting description

Tools and restrictions Use basic single inheritance, method overriding and appropriate use of `super()` to reuse initialization and description behavior. No additional libraries or exceptions are required.

Reference example

```
Provided base class:
class LibraryItem:
    def __init__(self, title):
        self.title = title

    def describe(self):
        return 'Item: ' + self.title

Example usage:
book_history = AudioBook('History', 75)
print(book_history.describe())

Expected output:
Item: History (75 min)
```